OPEN TEXT
CORPORATION

PatMotif SGML Functions: Installation Notes

June, 1993

1.0 Background Information on SGML

In order to maximize Open Text's SGML functionality, it is necessary to be aware of SGML and its capabilities with respect to your documents. This section will introduce some of the terminology that is used in the rest of the documentation. The characteristics and function of SGML tags will be described along with the configuration files required by Open Text software to utilize SGML functionality.

1.1 SGML and structure elements

Standard Generalized Markup Language (SGML) is a system that allows you to "mark up" text with special tags. These tags specify the structure of the document you are working with. For instance, if you were writing a book, you would use special tags to say "this block of text is a paragraph", or "this block of text is a chapter title". And you can combine these tags: "this block of text, made up of a chapter title and one or more paragraphs, is a chapter". Saying that "this block of text is a paragraph" means that the paragraph is a structure element in your document. And saying that "this block of text, made up of a chapter title and one or more paragraphs, is a chapter" shows you that structure elements can be made of combinations of other structure elements.

1.2 SGML tags

In order to tell programs such as Open Text's Pat search engine that, for instance, "this block of text is a paragraph", you have to surround that block of text by tags. These tags usually exist in pairs: start-tags and end-tags. For the paragraph example, you would say that <P> is the start-tag. This tells the program that the text following the tag is to be considered part of a paragraph. To end the paragraph, you would use the end-tag </P>. The end-tag should have exactly the same name as the start-tag, except that the name inside the angle brackets should be preceded by a slash (/) character. The start- and end- tag pairs allow you and the program to quickly and easily see the structure elements in a document.

1.3 SGML document type

You can keep combining these structure elements until you get a single element that is made of combinations of the others. This element is called the document type. For instance, if you were writing a book, your document type would be BOOK. For a newspaper, you would have a document type called NEWPAPER.

1.4 SGML files

In order for programs such as PatMotif to be able to take advantage of the SGML tag information, you have to tell the program first that you have a document type and then how the different structure elements can be combined to form the document. Telling PatMotif that it has a document type is called declaring the document type and the file that does this is therefore called the document type declaration file. This file has a .inp extension. The file that defines how the different structure elements of your document fit together is called the document type definition file and has a .dtd extension. Your SGML tagged text, your actual document, is called the SGML document and has a .sgm extension.

In this section you have learned what SGML is and some of the terminology used when describing SGML documents. You know that SGML documents have tags describing the different structure elements and that these are combined to form the base document type. You have also been introduced to the files that make up the SGML document environment.

2.0 The SGML document environment

Your SGML document is made up of different structure elements (e.g. paragraphs, chapters, etc.) which are combined to form the base document type. To take advantage of the information contained in your SGML tagged document, you need to tell programs such as Open Text's Pat search engine first that you have a base document type and then how that document type is constructed of the different structure elements. In other words, the allowable fields (elements) have to be defined and legal nestings must be unambiguously declared. This requires two files: the .inp file declares the document type and tells the program where to find the .dtd file. The .dtd (document type definition) file defines the document structure elements (fields) and their allowable nestings. This section will give examples of both, as well as an example of SGML tagged text, all of which will be based on the example of a newspaper.

2.1 The Document Type Declaration (.inp) file

In this section, we will describe the function of the document type declaration file, describe its syntax, and give an example of it.

The document type declaration (.inp) file declares the name of the base document type for your SGML tagged document and describes in which file to find the formal definition of the document structure. The file usually only has one line, which has the following syntax:

```
<!DOCTYPE doctype SYSTEM "filename.dtd">
```

where `doctype` is the name of the base document type, and `filename.dtd` is the name of the file that contains the formal definition of the document structure (called the document type definition or DTD). The filename.dtd will be assumed to be in the current directory or in the directory specified by the environment variable SGMLREGION_PATH. Documents can be of two types: `SYSTEM` and `PUBLIC`. `PUBLIC` entries are those known to more than system, whereas `SYSTEM` documents are those that are specific to the system on which they are prepared. Our example is not going to refer to any other documents, so we can specify a `SYSTEM` entry.

For our newspaper example, we will call the doctype `NEWPAPER` and the `filename.dtd` file will be called `newpaper.dtd`, which we will assume exists in the current directory. Our document type declaration file, called `newpaper.inp`, now looks like this:

```
<!DOCTYPE NEWPAPER SYSTEM "newpaper.dtd">
```

In this section, we have described the function and syntax of the document type declaration (.inp) file. We have also constructed the first of the files needed for our newspaper example of an SGML environment.

2.2 The Document Type Definition (.dtd) file

This section gives an overview of the function and syntax of the document type definition (.dtd) file, and gives an example of its use. If you need further information, please refer to the books listed in Appendix 2.

The document type definition (called the DTD for the document) formally defines the structure of an SGML document, as well as the relationships between the different structure elements. It describes how simple structure elements, made up of characters, can be combined to form more complex structure elements, including the base document type. For the newspaper example, we can assume a newspaper is made up of stories, illustrations, and ads. Stories may be made up of paragraphs, a date, a byline, an author, and other pieces. Paragraphs may be made up only of text (character data), not other structure elements.

To construct our DTD, we will start with the topmost element of an SGML document, the base document type itself. Our example is for our newspaper example.

```
<!ELEMENT       NEWPAPER        O O     (STORY)*>
```

The element NEWPAPER is made up of zero or more STORYs. The * means zero or more. The first of the capital letter O's means that the start-tag for this structure element can be omitted from the actual SGML document. The second capital letter O means that the end-tag for can be omitted.

The next step is to define what a STORY element looks like:

```
<!ELEMENT       STORY           - -     (TEXT|ILLUST)*>
```

For our newspaper example, a story is made up of zero or more TEXTs or ILLUSTrations. The vertical bar (|) means "or". The dashes (-) mean that the start- and end-tags must be present in the text in order for the structure element to be recognized as a paragraph. We can also say that the STORY has certain attributes, such as STATUS, PUBlisher, PAGE, DATE, etc, and each of these attributes can have a value associated with it. So, for our newspaper example, we add the following:

```
<!ATTLIST     STORY
              STATUS     (Draft|Prepare|Ready) Draft
              PUB        CDATA                 "Local Newspaper"
              DATE       NUMBER                #IMPLIED
              PAGE       CDATA                 "  ">
```

This tells us that the structure element STORY has an ATTribute LIST that includes STATUS, PUBlisher, DATE, and PAGE. The STATUS attribute can take one of three values (`Draft`, `Prepare`, or `Ready`), but defaults to `Draft`. The `PUB`lisher attribute is made up of characters (`CDATA`) and defaults to "Local Newspaper". The `DATE` attribute is a number. The `#IMPLIED` tells the system that there is no default for the attribute, but that the system should imply a value if none is given. The `PAGE` is simply character data and defaults to a blank.

`TEXT` and `ILLUST`ration elements are, for the purposes of our newspaper example, made up of characters. To be recognized, they will require both start- and end- tags. To enter this into the DTD, we write:

```
<!ELEMENT       TEXT          - -     CDATA>
<!ELEMENT       ILLUST        - -     CDATA>
```

By looking at all the entries together, it is easy to see that our base document type `NEWPAPER` is made up of zero or more `STORY`s. A `STORY` is made up of zero or more pieces of `TEXT` or `ILLUST`rations, each of which are in turn made up of characters. Each `STORY` also has a `STATUS`, `PUB`lisher, `DATE`, and `PAGE` associated with it. The following puts all the information together. The <-- and --> tags are regarded as comments.

```
<!-- This example .dtd file should be called newpaper.dtd -->
<!ELEMENT       NEWPAPER      O O     (STORY)*>
<!ELEMENT       STORY         - -     (TEXT|ILLUST)*>
<!ATTLIST       STORY
                STATUS(Draft|Prepare|Ready)   Draft
                PUBCDATA                     "Local Newspaper"
                DATENUMBER                    #IMPLIED
                PAGECDATA                     " ">
<!ELEMENT       TEXT          - -     CDATA>
<!ELEMENT       ILLUST        - -     CDATA>
```

This section has taught us that the function of the document type definition (.dtd) file is to define the relationships amongst the different structure elements that make up an SGML document. We have also shown examples of its syntax in constructing the `newpaper.dtd` file we will use for our newspaper example of an SGML document.

2.3 The SGML document (.sgm) file

The actual SGML tagged text is stored in this file and the DTD (stored in the document type definition (.dtd) file) is used to interpret the text and its various elements (fields). Notice that the elements defined in the DTD are called tags when surrounded by angle brackets (<>). Also notice that the same line that appeared in the document type declaration (.inp) file is also the first line of the SGML (.sgm) document file. The attributes of the STORY element and their associated values are all contained within the STORY tag as well. The following is some sample text which, in this example, would be stored in the file called newpaper.sgm:

```
<!DOCTYPE NEWPAPER SYSTEM "newpaper.dtd">

<STORY STATUS="Draft" PUB="Local Newspaper" DATE="930520"
PAGE="A1">

<TEXT>WATERLOO (OT) -- Open Text Corporation, the large scale
text management systems company that has always supported the
SGML standard, now has a new parallel text search engine. </TEXT>

</STORY>

<STORY STATUS="Ready" PUB="Local Newspaper" DATE="930520"
PAGE="A1">

<ILLUST>File Photo/ The Open Text Corporation President </ILLUST>

</STORY>
```

2.4 The SGML document environment

This section reviewed the concepts of SGML and that the base document type of your SGML document is made up of a number of different structure elements, which may themselves be made up of other structure elements. The base document type should be first declared in the document type declaration (.inp) file. The function of the document type definition (.dtd) file is then to describe the relationships amongst the different structure elements of your document, as well as to describe the attributes that are associated with these elements. The SGML (.sgm) file is itself the actual SGML tagged document whose structure is defined with the first two documents.

3.0 SYSTEM SETUP

This section will take you through the steps required to set up your SGML tagged document for use with the SGML functions of PatMotif. We will introduce the configuration file requirements and check the correctness of your SGML document. The benefits of using a regions file for your document will be introduced and explained, and we will initialize a default region. We will also learn how to easily have a simple Lector specifications file generated for us. The benefits of an SGML filter between PatMotif and Lector will also be described. When these steps are completed, you will be ready to use the SGML functions of PatMotif.

The many different modes of the `sgmlregion` program will be used throughout this section, so an explanation of each of the modes used will be given here. The different modes are selected by using them as arguments to the -m command line option to `sgmlregion`. Please refer to the man pages on `sgmlregion` for a fuller discussion.

Mode	Function
check	This mode validates the DTD contained in the .dtd file and the SGML document itself. `sgmlregion` will report any syntax or other errors.
filter	This mode gets `sgmlregion` to parse the DTD and wait for standard input.
region	-D data_dictionary_name.dd This mode generates all the regions in the file and updates the region information in the data dictionary file. The -D must be included to specify which data dictionary file is to be updated. The name of region file created will be the same as that of the text with a .P extension.
root	This mode determines and prints out the root element (also referred to as the base document type for our purposes) of the SGML document. This is then used in the initialization (.ini) file for PatMotif.
spec	This mode will generate a simple Lector specifications (.spc) file, where all elements are recorded. See the man pages for Lector or Chapter 4 Configuring Lector of the Installation Guide for a fuller discussion of the specifications file.

It is important to note why the `sgmlregion` program is used here, instead of using the `patregion` or `multiregion` programs. The `patregion` program was designed to be used when the document has arbitrary tags to denote regions. The `multiregion` program was designed to be used when the document had normal SGML type tags (with angle (<>) brackets), but no DTD. The `sgmlregion` program was designed to be used with fully validated SGML documents with a DTD. This can be summarized with the following table.

Region Builder	Appropriate Use
`patregion`	text has arbitrary tags
`multiregion`	text has SGML type tags but no DTD
`sgmlregion`	text has fully validated SGML tags and a DTD

Throughout this section, reference will be made to the files `newpaper.inp`, `newpaper.dtd`, and `newpaper.sgm`. These are the names of the example files described in the section entitled "The SGML document environment".

3.1 Configuration File Requirements

This subsection will describe to you the three files that are necessary to use the SGML functions in PatMotif. These files are in addition to the PatMotif start-up file required for PatMotif and the data dictionary file required by Open Text's Pat search engine. Instructions will be given to guide you in creating a simple data dictionary file should this be necessary. An example PatMotif start-up file is shown in Appendix 1.

The three files described in the section entitled "The SGML document environment" are necessary to use the SGML functions of PatMotif. These files are the document declaration file (with a .inp extension), and the document definition file (with a .dtd extension), which together describe the structure of the actual SGML tagged text in the .sgm file.

In addition to these, you will require the files that Pat and PatMotif needs to function without using the SGML functions. These are the data dictionary (required for Pat) and PatMotif start-up files. The data dictionary file contains most of the information necessary for Pat and PatMotif to function, including character mappings, stopwords, indexes, regions, and the locations of their respective files. To create a simple default data dictionary, you can use the patbld program as follows:

```
patbld -v -m memory_size -t text_name -o output_name
```

The -v option makes the output verbose, meaning that the `patbld` program will describe to you each step that it is taking along with timing statistics for its operation. The -m option describes the amount of memory to be used for the `patbld` process (for a further discussion of this, please refer to the sections 2.3 Building the Index and 2.4 Choosing the Parameters for patbld from the Installation Guide). The SGML tagged document is given as the text name and the -o specifies what the output files are to be called. For the newspaper example, we would use `patbld` with the following options:

```
patbld -v -m 1M -t newpaper.sgm -o newpaper
```

This will give `patbld` 1 megabyte of memory to build its indices. The results of the `patbld` operation are the files `newpaper.dd` and `newpaper.idx`. The .dd file is the data dictionary file, and the .idx file is the index file.

There is a sample PatMotif start-up file in Appendix 1. It tells PatMotif what program to run, what help file to use, and what output options are available to it. For a fuller discussion of the PatMotif start-up file, please refer to section 3.1.1.4, PatMotif Control File, in the Installation Guide.

In this subsection, we have described the need for the SGML files as well as the data dictionary and PatMotif start-up files. These must be available before the next step can be taken.

3.2 Checking the SGML document correctness

In order to ensure that there are no syntax or other errors in the DTD (contained in the .dtd file) or the SGML tagged document (in the .sgm file) itself, we have to run a test over them. The `sgmlregion` program provides us with an easy way to perform this checking process: we will use the `check` mode. The -v option again makes the output verbose, with `sgmlregion` telling what it is doing as it does it. The -m option selects the mode (see above and the man pages under `sgmlregion`). Substitute your SGML file for the newpaper.sgm file given here.

```
sgmlregion -v -m check newpaper.sgm
```

Please note that all fatal errors must be overcome before PatMotif will be able to use the document. If successful, messages similar to the following should be returned:

```
check mode ...

checking total size(0K) time (0s)
```

With this simple step, we now know that our SGML document and DTD are correct and fully validated. We can now move on to building the regions file.

3.3 Building the SGML regions

One of the benefits associated with using SGML documents is that we can define regions of text. A region is the text that exists between the start- and end-tags of an SGML document structure element. So, for instance, text between <P> and </P> tags will be referred to as a region. When lists of regions are placed into a regions file, programs such as Pat are given the added functionality that they can restrict searches to only paragraphs (for example), or some other defined region. We must remember, however, that we must already have a data dictionary file before we try to build our regions file, as the region builder will update the data dictionary file with the new region information. The SGML region builder is invoked with the following command:

```
sgmlregion -v -m region -D newpaper.dd newpaper.sgm
```

You can substitute your data dictionary name for the `newpaper.dd` file and your SGML document for the `newpaper.sgm` file. The region information is derived from the actual SGML document, and the results placed in the regions file. For our example, the regions file would be called `newpaper.P`. If the region building operation is successful, then messages similar to the following will be displayed (the number of regions, their sizes, and the time shown are for the example given):

```
region mode ...
.. building regions#(0) size(0K) time(0s)
total built regions#(18) size(0K) time(0s)
.. sorting now
.. writing now
```

In addition to these messages, for each region built a message similar to the following will be given:

```
built (newpaper.P) region (NEWPAPER                    count=     2)
```

It is important to note that if the data dictionary (.dd) file already contains Regions information between <Regions> and </Regions> tags, this information must be deleted to ensure that the new region information is properly updated.

If, for some reason, a region is not needed or not wanted for Pat or PatMotif, it has to be manually deleted from the data dictionary after the regions building process is completed. For instance, if `ILLUST`rations are not needed, the information between the

```
  <Region>
      <Name>ILLUST</Name>
         ...
  </Region>
```

tags can be deleted from the data dictionary, thereby making the `ILLUST` region invisible to Pat or PatMotif.

For a discussion of why `sgmlregion` is used here instead of either the `patregion` or `multiregion` programs, please refer to the discussion (above) that opens this section.

In this subsection, we have described what regions are, how to build them and what benefits we derive from doing so. We have also described how to remove unnecessary or unwanted regions from the data dictionary should we decide to do so. We can now move on to determining our default region.

3.4 Initializing the default regions

Regions are simple but powerful tools for restricting searches. With them, we can restrict our searches so that we are only searching in headlines, or only in section titles, or only in some other defined region. Regions also allow us the flexibility to control how much of a text is returned for display when a search is successful and we wish to display the results.

It is easy for us to dynamically change how much of a text to display when using PatMotif (see section 4.3 of the PatMotif Tutorial, Displaying According to Compnent, for details on how to do this). However, we need to have a specify a default region to allow us to simply search without first specifying what region to use. The root mode of sgmlregion allows us to easily determine a suitable default region for display and will automatically generate the Pat command that will set this default region for us. In fact, the sgmlregion program uses the base document type, the topmost element of the DTD, as the default region.

We can easily place the Pat command generated by the sgmlregion program to set the default region into a Pat initialization (.ini) file. (For a broader discussion of the Pat initialization file, please refer to the Installation Guide) This can be done with the following simple command:

```
sgmlregion -m root newpaper.inp > newpaper.ini
```

For our newspaper example, `newpaper.inp` is the document type declaration file and `newpaper.ini` is the generated initialization file. The `newpaper.ini` file will consist of only the following line:

```
{DefaultRegion NEWPAPER}
```

We now have a Pat initialization file. Now that we have the file, we have to inform Pat that it should read and execute this file when it starts up. To do this, we simply have to add the following line to the data dictionary file. We will add it just after the </File> tag in the first set of <Index> and </Index> tags. So, for our example, we would have

```
 ...

</File>

<InitFile>newpaper.ini</InitFile>

<IndexPoints>
```

. . .

While the previous subsection discussed the benefits and uses of the region information, this subsection has shown how to use the root mode of the sgmlregion program to automatically determine the default region for your document. The output of the sgmlregion program, a valid Pat command to set the default region, was placed in the Pat initialization file. We then placed a single line into our data dictionary to inform Pat of our newly created initialization file.

3.5 Generating a simple Lector specifications (.spc) file

For a fuller discussion of Open Text's Lector text browser and the flexibility in display provided through its specifications file, please refer to Chapter 4, Configuring Lector, in the Installation Guide.

The powerful `sgmlregion` program will be used here again to automatically generate a simple Lector specifications file, which we can then easily modify to suit our needs using Open Text's Lector Spec File Editor. We will also explain the difference between the Standard and View Tags display formats generated for us in this generated specifications file.

To generate a simple specifications file for use with Lector, we will use the `spec` mode of `sgmlregion`. It will search through the text, collecting the names of all the tags used, and then generate a specifications file with each tag listed but no display action specified. It will also generate two default formats with which you can view the text. The Standard format will not display any tags, only the text of your document. The View Tags format will display both the tags and the text in your document.

To have `sgmlregion` generate the specifications file, invoke it with the following command:

```
sgmlregion -m spec newpaper.inp > newpaper.spc
```

or, alternatively:

```
sgmlregion -m spec newpaper.sgm > newpaper.spc
```

The resulting specifications file is called `newpaper.spc`.

Should you wish to change what actions are associated with the tags, you can do so either by using Open Text's Lector Spec File Editor, or by editing the specifications file directly.

This subsection has introduced you to Lector, Open Text's text browser. We have described what it does and how it works by performing certain actions when it encounters different tags in the text document. We have used the `sgmlregion` program to automatically generate a simple specifications file for us, which we can then edit either directly or with Open Text's Lector Spec File Editor.

3.6 Updating the PatMotif start up (.pat) file to include the SGML filter

This subsection describes the function of the PatMotif start up (.pat) file and explains what benefits are derived by using the SGML `filter` mode of `sgmlregion`. It then shows you how to modify the .pat file to invoke the filter.

The function of the PatMotif start up (.pat) file, as described in the introduction to this section, is to tell PatMotif what program to run, what help file to use, and what output options are available to it. For a broader discussion of the PatMotif start-up file, please refer to section 3.1.1.4. PatMotif Control File in the Installation Guide.

The filter mode of the `sgmlregion` program will read in the DTD information from the document declaration file. It will then use that information to normalize the tagged text going to Lector from Pat. Because the DTD allows you to omit start- or end-tags, it can be difficult for Lector to determine where one structure element ends and where another begins. The filter will therefore add any missing tags or change the incoming information in such a way as to ensure that Lector does not become confused. This may also involve expanding entity references.

To show you how and where to add the filter you will need to refer to the start up file in Appendix 1. The command given between the <Command> and </Command> tags was changed from

```
LectorMotif -g =480x640-20-20 -N -C 2 -F newpaper.spc
```

to

```
sgmlregion -m filter newpaper.inp - | LectorMotif -g =480x640-20-
20 -N -C 2 -F newpaper.spc
```

The dash at the end of the `sgmlregion -m filter newpaper.inp -` command means that the sgmlregion program will wait for information on the standard input. The vertical bar (|) character is called a pipe and means that the expanded text will be passed onto the Lector text browser for display.

This subsection has described the function of the PatMotif start up file and of the SGML filter mode of `sgmlregion`, and has shown you how to modify the .pat file to invoke the filter. You are now ready to start using PatMotif with the SGML functions.

3.7 Using PatMotif with SGML functions

With all the necessary preparations completed as described above, you are now ready to start using PatMotif and its SGML functions. To do so, you should invoke PatMotif with the following command:

```
PatMotif.sgml -SGML newpaper.inp newpaper.pat
```

The name of your document declaration file should be substituted for the `newpaper.inp` file and your PatMotif start up file should be substituted for the `newpaper.pat` file. The -SGML flag to PatMotif indicates that you want to use the SGML functions and that the DTD for your SGML document is in the .inp file.

3.8 Final Installation Notes:

In order to use PatMotif, the necessary X11 resources have to be made available. This means that the `Lector.ad` and `PatMotif.ad` resources should be merged with the X Resource DataBase.

3.9 Conclusion

This section described the steps required to set up your SGML tagged document for use with the SGML functions of PatMotif. First, we examined the need for two extra configuration files. We then checked the SGML document for correctness. We used the sgmlregion program to automatically generate a regions file for our document, and then to determine the default region. We placed the generated Pat command to set the default region into a Pat initialization file. We also used the sgmlregion program to generate a simple Lector specifications file for us. We then discussed the need to use a SGML filter between Pat and Lector. You should now be ready to use the SGML functions of PatMotif. Should you require help to do this, please refer to the User Notes for PatMotif SGML Functions.

4.0 Appendix 1

A Sample PatMotif start up (.pat) file

```
<Program>pat4.0 newpaper.dd</Program>
<Help>/usr/ot/demos/X11-supp/PatMotif.help</Help>
<Routing>
  <Output><Label>Lector</Label>
         <Command>sgmlregion -m filter newpaper.inp - |
LectorMotif -g =480x640-20-20 -N -C 2 -F newpaper.spc</Command>
         <Data>Text</Data>
         <Help>2-Column Lector</Help>
  </Output>
  <Output><Label>dump</Label>
         <Command>cat</Command>
         <Data>Text</Data>
         <Help>Screen Dump</Help>
  </Output>
</Routing
```

5.0 Appendix 2

Suggested readings and reference materials on SGML

The SGML Handbook: The annotated full text of ISO 8879 - Standard Generalized Markup Language -- Dr. Charles F. Goldfarb. Claredon Press, Oxford, 1990

SGML: An Author's Guide to the Standard Generalized Markup Language -- Martin Bryan. Addison-Wesley Publishing Company, New York, 1988

SGML and Relate Standards: Document Description and Processing Language -- Joan Smith. Ellis Horwood, New York, 1992